BDM PROJECT

Team Members

Parul Ghai
Prateek Gupta
Priyal Katudia
Shubham Bhargava

Tasks and Accountability contract:

Work Breakdown	Parul Ghai	Prateek Gupta	Priyal Katudia	Shubham Bhargava
Search and create datasets	•	•		
ER Diagram	•			
Create SQL code and import datasets to database	•	•	•	•
Dataset cleaning with SQL code		•		
Design your own SQL question and	•	•	•	•

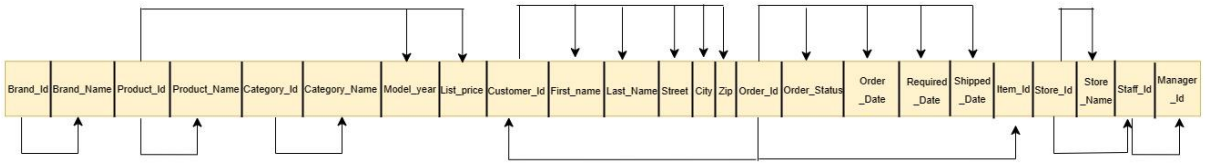
write SQL queries				
Presentation	●	●	●	●
Visualizations (Optional)	●	●	●	●

Dataset

The Bike Store SQL dataset project involves designing and implementing a relational database to manage data for a fictional bike store. The dataset comprises eight interconnected tables: customers, orders, order_items, staffs, stores, categories, products, and brands. This project aims to provide a comprehensive solution for tracking customer interactions, sales transactions, inventory management, and employee activities within the bike store.

Data Normalization

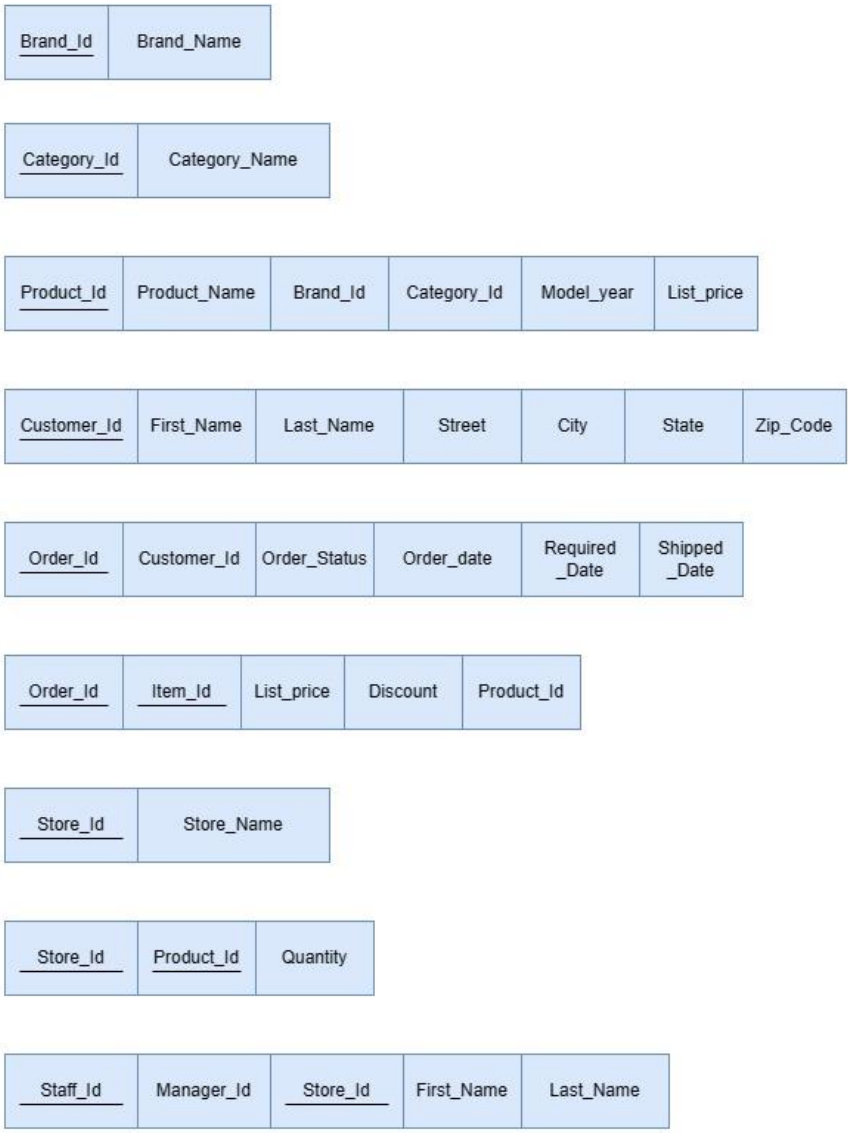
Raw data before normalization:



- The data is redundant and has created the following anomalies:
  - **Creation Anomalies:**
    - If a new customer is added, and the order Id is not already present in the respective tables, then there would be a creation anomaly.

- **Updation Anomalies:**
  - If the name of a store is updated, then all the records that reference that store name would need to be updated to avoid inconsistency.
- **Deletion Anomalies:**
  - If the list price of the product is deleted, then all the records related to the product would be deleted, which would result in losing crucial information on other columns.

Normalized data structure:



- **Customers Table:**
    - Customer_ID (Primary Key)
    - First_Name
    - Last_Name
    - Street
    - City
    - State
    - Zip

- **2. Orders Table:**
    - Order_ID (Primary Key)
    - Customer_ID (Foreign Key)
    - Order_Date
    - Order_Status
    - Required_Date
    - Shipped_Date

- **Order_Items Table:**
    - Order_ID (Foreign Key)
    - Item_ID (Foreign Key)
    - List_price
    - Dicount
    - Product_Id

- **Staffs Table:**
    - Staff_ID (Primary Key)
    - Store_Id (Primary Key)
    - Manager_Id
    - First_Name
    - Last_Name

- **Stores Table:**
    - Store_ID (Primary Key)
    - Product_Id(Primary Key)
    - Quantity

- **Categories Table:**
    - Category_ID (Primary Key)
    - Category_Name

- **Products Table:**
    - Product_ID (Primary Key)
    - Product_Name
    - Brand_Id
    - Category_ID (Foreign Key)
    - Model_Year
    - List_Price

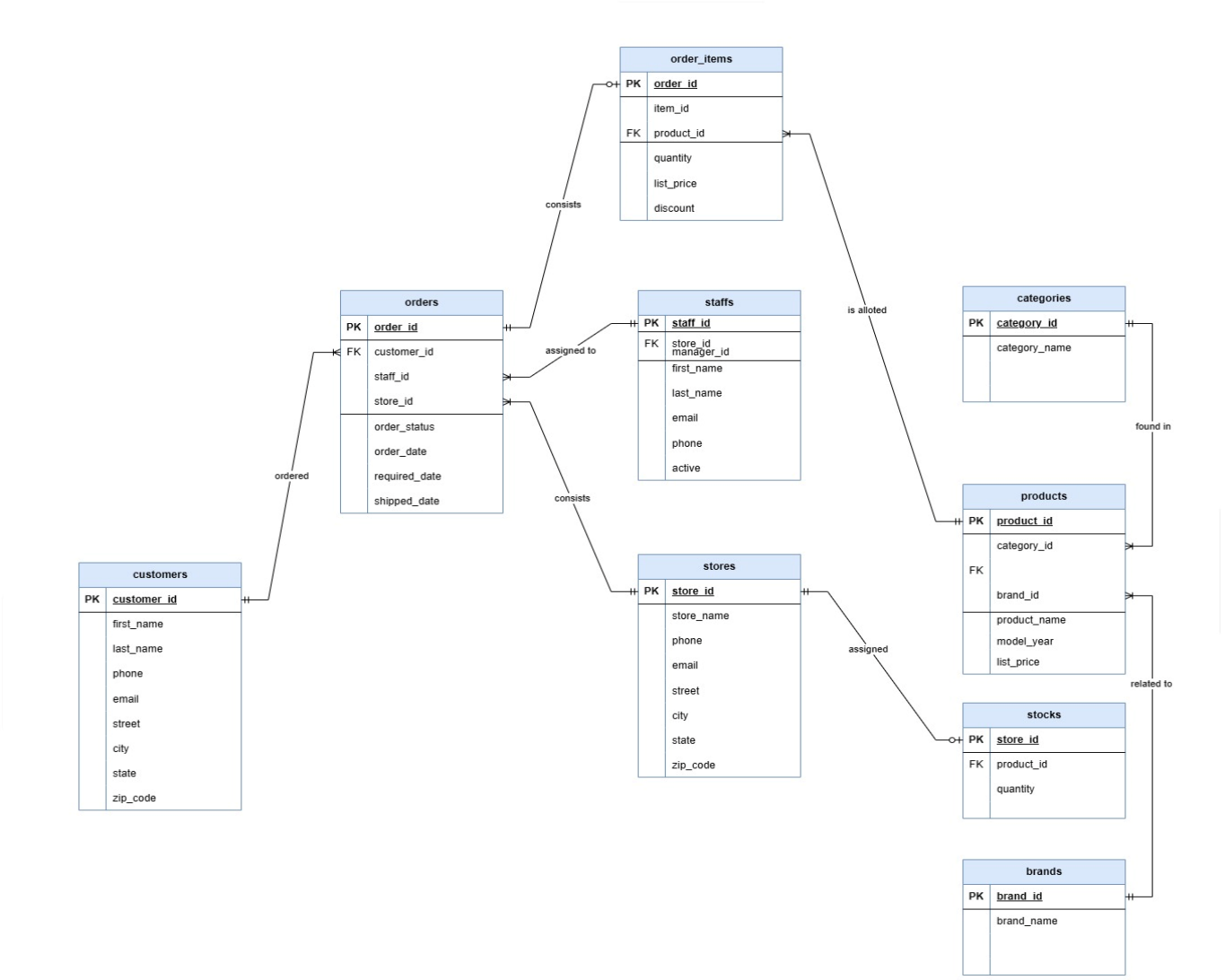
- **Brands Table:**
    - Brand_ID (Primary Key)
    - Brand_Name

Entity-Relation Diagram

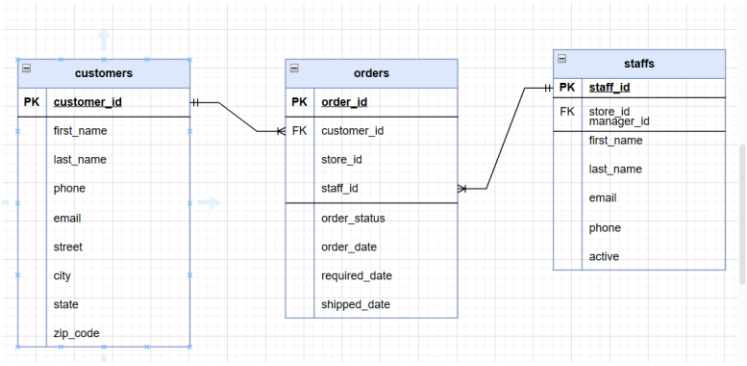
Business Rules:

- The Customers table is linked to the Orders table via the Customer_ID field, establishing a one-to-many relationship, as a customer can place multiple orders.
- The Orders table is connected to the Order_Items table through the Order_ID, allowing a detailed breakdown of items within each order.
- The Order_Items table references both the Orders and Products tables to associate each item with a specific order and product.
- The Staffs and Stores tables provide additional context, linking staff members to specific store locations through order_id from orders table.

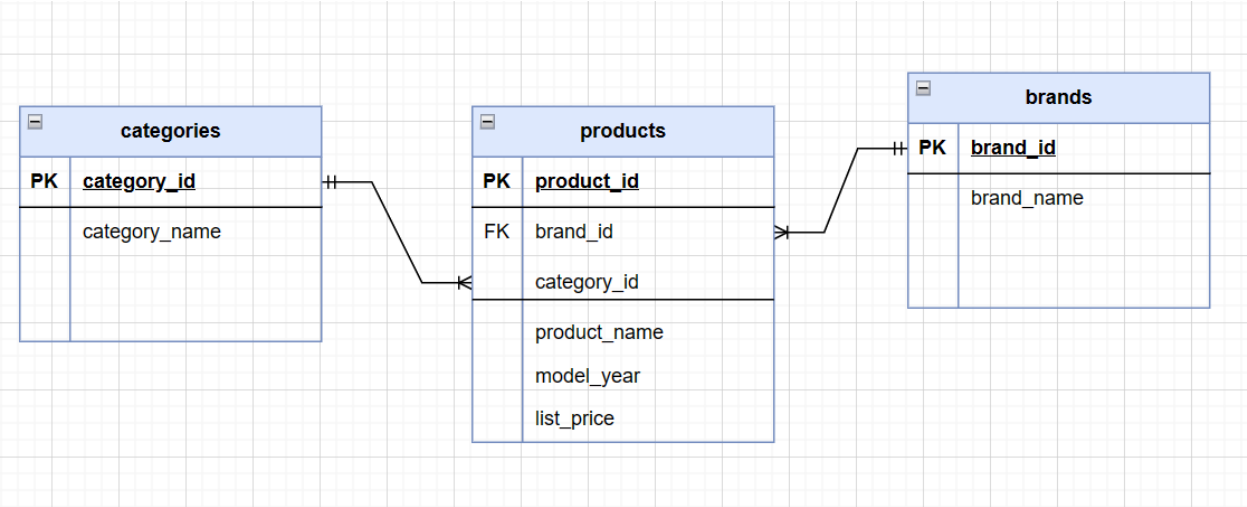
ER Diagram



Customer and Staff have many to many relationship which is bridged by the Orders table.



Categories and Brands have many to many relationship which is bridged by the Product table.



Data Cleaning

Modify column:

ALTER TABLE your_table
ALTER COLUMN column_name new_data_type;

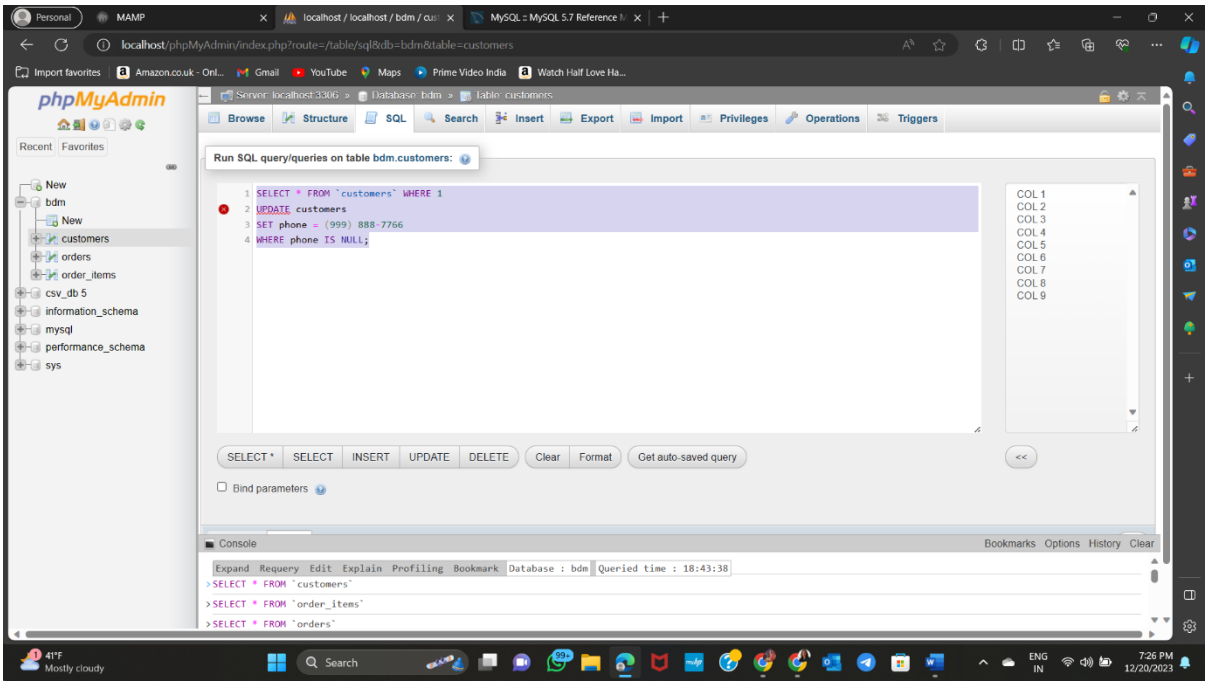
Renamed all the columns using query:

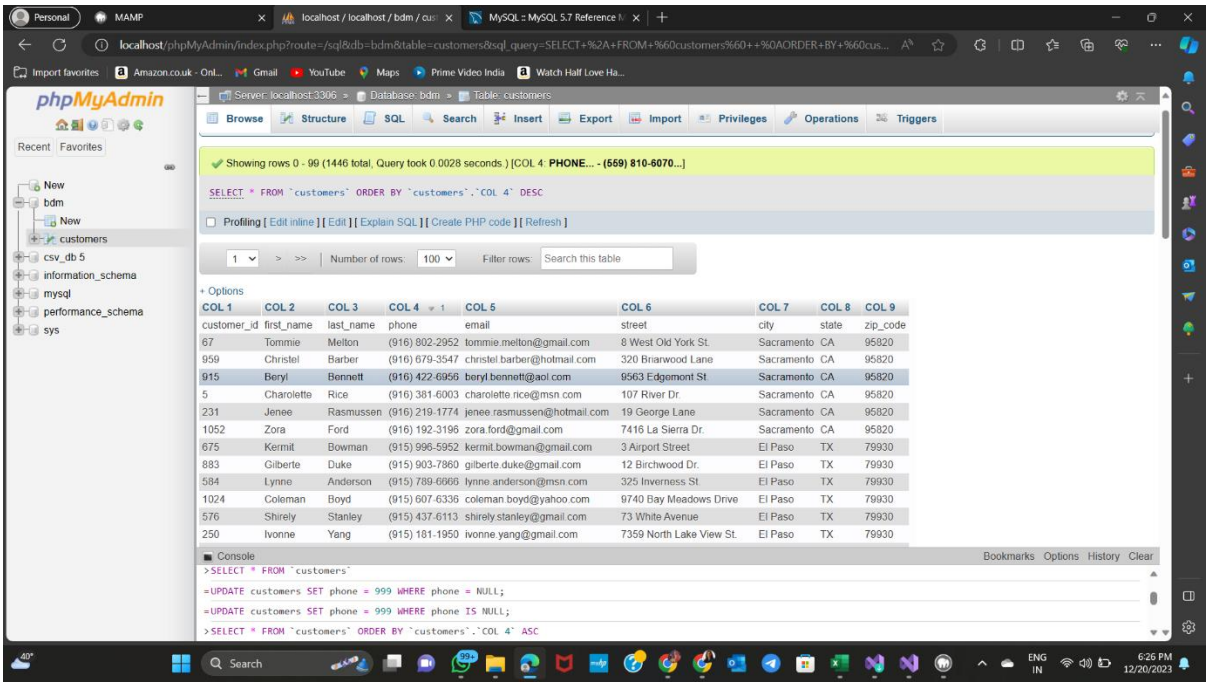
ALTER TABLE your_table
RENAME COLUMN old_name TO new_name;

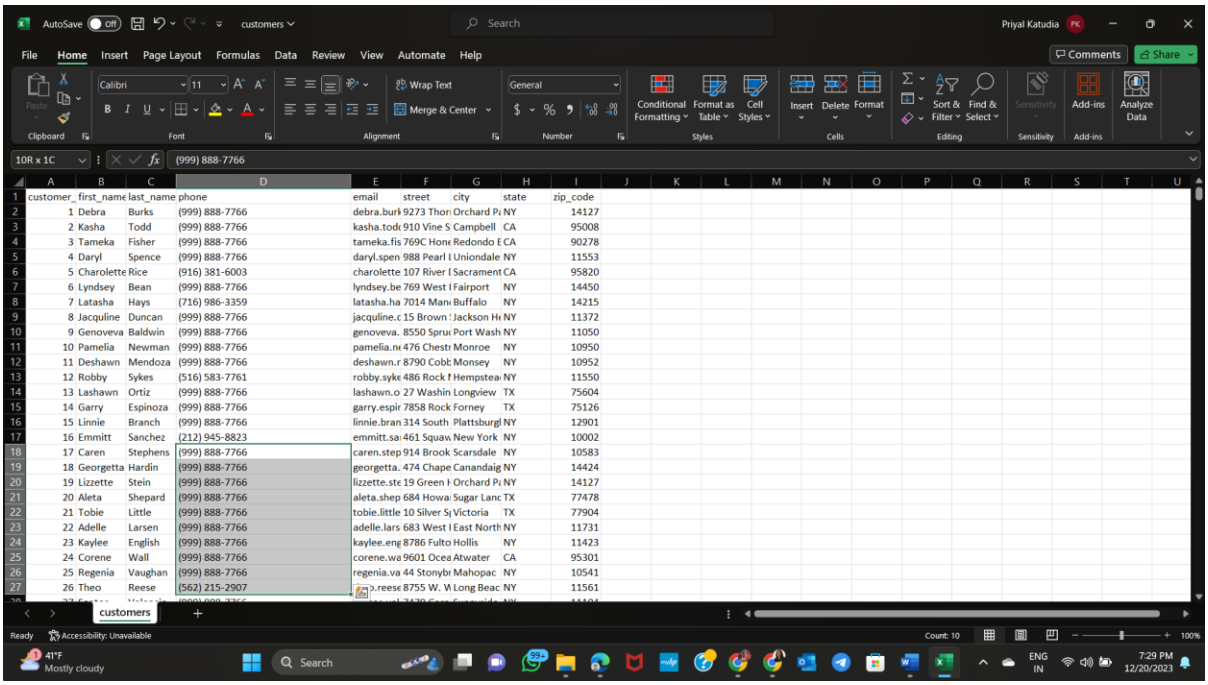
Handle missing data-

Replace missing values:

SELECT * FROM `customers` WHERE 1
UPDATE customers
SET phone = (999) 888-7766
WHERE phone IS NULL;


phpMyAdmin
Server: localhost:3306 » Database: bdm » Table: customers
Browse Structure SQL Search Insert Export Import Privileges Operations Triggers
Run SQL query/queries on table bdm.customers:
1 SELECT * FROM `customers` WHERE 1
2 UPDATE customers
3 SET phone = (999) 888-7766
4 WHERE phone IS NULL;
COL 1
COL 2
COL 3
COL 4
COL 5
COL 6
COL 7
COL 8
COL 9
SELECT * SELECT INSERT UPDATE DELETE Clear Format Get auto-saved query
Bind parameters
Console
Bookmarks Options History Clear
Expand Requery Edit Explain Profiling Bookmark Database : bdm Queried time : 18:43:38
>SELECT * FROM `customers`
>SELECT * FROM `order_items`
>SELECT * FROM `orders`

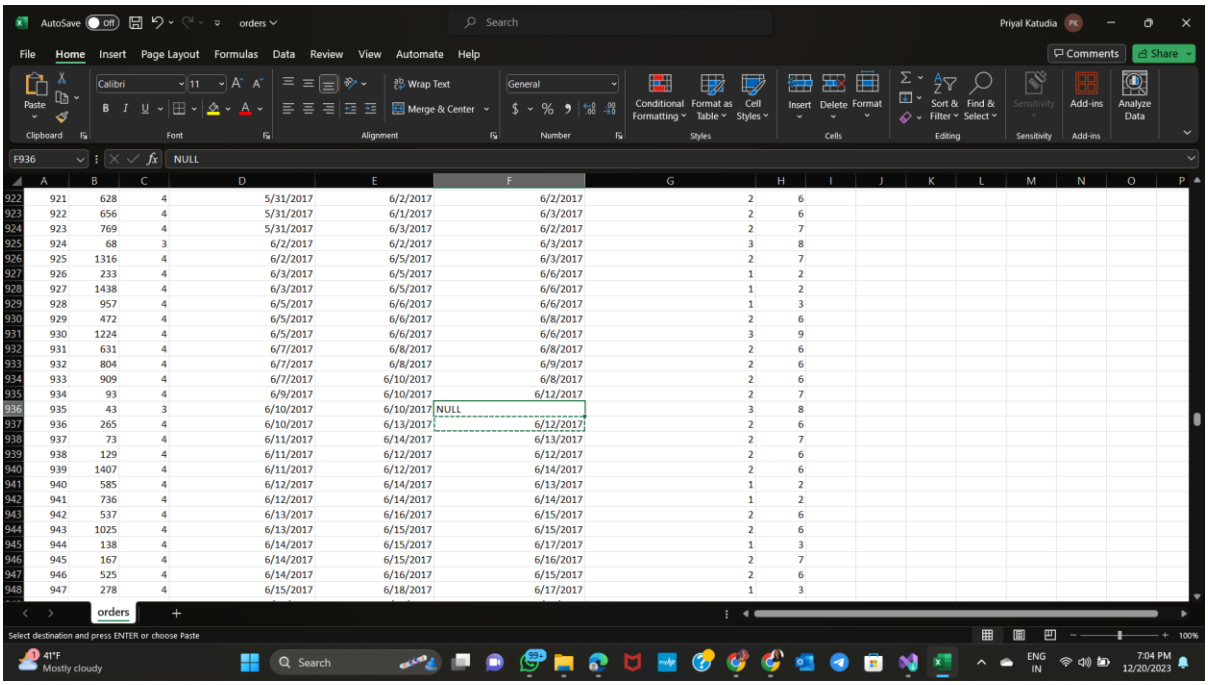

phpMyAdmin
Server: localhost:3306 » Database: bdm » Table: customers
Browse Structure SQL Search Insert Export Import Privileges Operations Triggers
Showing rows 0 - 99 (1446 total, Query took 0.0028 seconds.) [COL 4: PHONE... - (659) 810-6070...]
SELECT * FROM `customers` ORDER BY `customers`.`COL 4` DESC
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Number of rows: 100 Filter rows: Search this table
+ Options
COL 1 COL 2 COL 3 COL 4 COL 5 COL 6 COL 7 COL 8 COL 9
customer_id first_name last_name phone email street city state zip_code
67 Tommie Melton (916) 802-2952 tommie.melton@gmail.com 8 West Old York St. Sacramento CA 95820
959 Christel Barber (916) 679-3547 christel.barber@hotmail.com 320 Briarwood Lane Sacramento CA 95820
915 Beryl Bennett (916) 422-6956 beryl.bennett@aol.com 9563 Edgemont St. Sacramento CA 95820
5 Charolette Rice (916) 381-6003 charolette.rice@msn.com 107 River Dr. Sacramento CA 95820
231 Jenee Rasmussen (916) 219-1774 jenee.rasmussen@hotmail.com 19 George Lane Sacramento CA 95820
1052 Zora Ford (916) 192-3196 zora.ford@gmail.com 7416 La Sierra Dr. Sacramento CA 95820
675 Kermit Bowman (915) 996-5952 kermit.bowman@gmail.com 3 Airport Street El Paso TX 79930
883 Gilberte Duke (915) 903-7860 gilberte.duke@gmail.com 12 Birchwood Dr. El Paso TX 79930
584 Lynne Anderson (915) 789-6666 lynne.anderson@msn.com 325 Inverness St. El Paso TX 79930
1024 Coleman Boyd (915) 607-6336 coleman.boyd@yahoo.com 9740 Bay Meadows Drive El Paso TX 79930
576 Shirely Stanley (915) 437-6113 shirely.stanley@gmail.com 73 White Avenue El Paso TX 79930
250 Ivonne Yang (915) 181-1950 ivonne.yang@gmail.com 7359 North Lake View St. El Paso TX 79930
Console
Bookmarks Options History Clear
>SELECT * FROM `customers`
=UPDATE customers SET phone = 999 WHERE phone = NULL;
=UPDATE customers SET phone = 999 WHERE phone IS NULL;
>SELECT * FROM `customers` ORDER BY `customers`.`COL 4` ASC



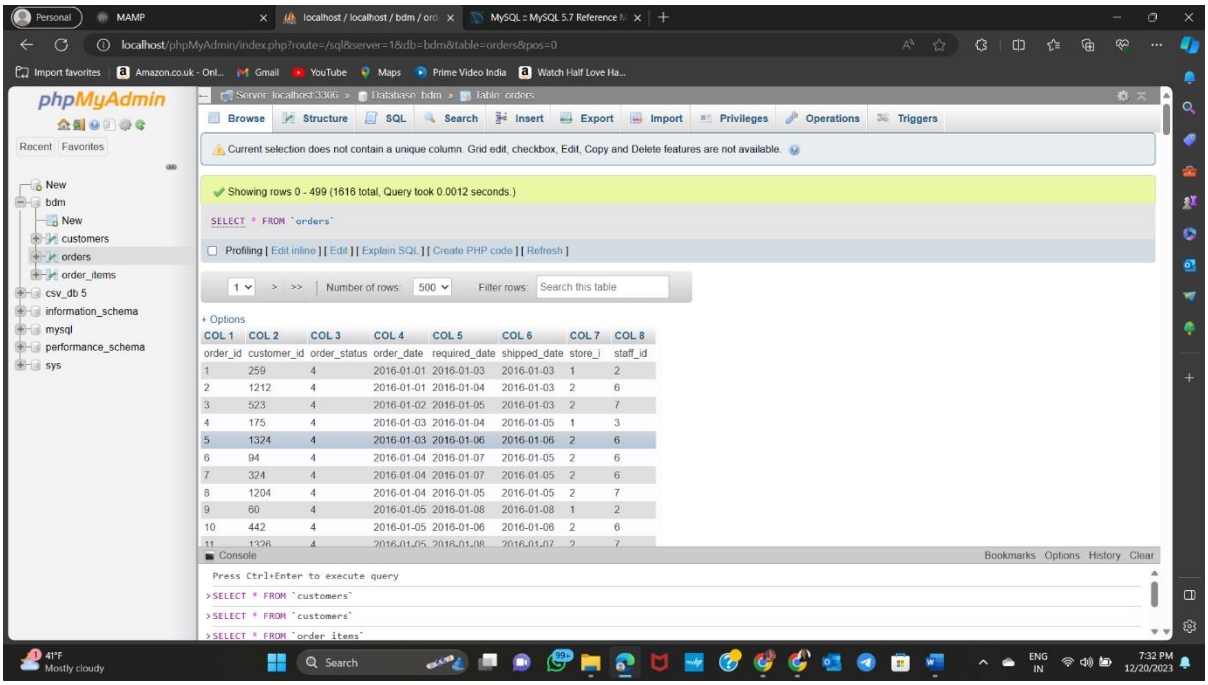
customer_	first_name	last_name	phone	email	street	city	state	zip_code
1	Debra	Burks	(999) 888-7766	debra.burk	9273 Thor	Orchard P	NY	14127
2	Kasha	Todd	(999) 888-7766	kasha.todd	910 Vine S	Campbell	CA	95008
3	Tameka	Fisher	(999) 888-7766	tameka.fis	769C Hone	Redondo B	CA	90278
4	Daryl	Spence	(999) 888-7766	daryl.spen	988 Pearl I	Uniondale	NY	11553
5	Charolette	Rice	(916) 381-6003	charolette	107 River I	Sacrament	CA	95820
6	Lyndsey	Bean	(999) 888-7766	lyndsey.be	769 West I	Fairport	NY	14450
7	Latasha	Hays	(716) 986-3359	latasha.ha	7014 Mand	Buffalo	NY	14215
8	Jacquline	Duncan	(999) 888-7766	jacquline.c	15 Brown :	Jackson He	NY	11372
9	Genoveva	Baldwin	(999) 888-7766	genoveva.	8550 Spruc	Port Wash	NY	11050
10	Pamelia	Newman	(999) 888-7766	pamelia.ne	476 Chestr	Monroe	NY	10950
11	Deshawn	Mendoza	(999) 888-7766	deshawn.r	8790 Cobb	Monsey	NY	10952
12	Robby	Sykes	(516) 583-7761	robby.syke	486 Rock I	Hempstead	NY	11550
13	Lashawn	Ortiz	(999) 888-7766	lashawn.o	27 Washin	Longview	TX	75604
14	Garry	Espinoza	(999) 888-7766	garry.espir	7858 Rock	Forney	TX	75126
15	Linnie	Branch	(999) 888-7766	linnie.bran	314 South	Plattsburgl	NY	12901
16	Emmitt	Sanchez	(212) 945-8823	emmitt.sa	461 Squaw	New York	NY	10002
17	Caren	Stephens	(999) 888-7766	caren.step	914 Brook	Scarsdale	NY	10583
18	Georgetta	Hardin	(999) 888-7766	georgetta.	474 Chape	Canandaig	NY	14424
19	Lizzette	Stein	(999) 888-7766	lizzette.ste	19 Green I	Orchard P	NY	14127
20	Aleta	Shepard	(999) 888-7766	aleta.shep	684 Howa	Sugar Lanc	TX	77478
21	Tobie	Little	(999) 888-7766	tobie.little	10 Silver S	Victoria	TX	77904
22	Adelle	Larsen	(999) 888-7766	adelle.lars	683 West I	East North	NY	11731
23	Kaylee	English	(999) 888-7766	kaylee.eng	8786 Fulto	Hollis	NY	11423
24	Corene	Wall	(999) 888-7766	corene.wa	9601 Ocea	Atwater	CA	95301
25	Regenia	Vaughan	(999) 888-7766	regenia.va	44 Stonybr	Mahopac	NY	10541
26	Theo	Reese	(562) 215-2907	o.reese	8755 W. W	Long Beac	NY	11561

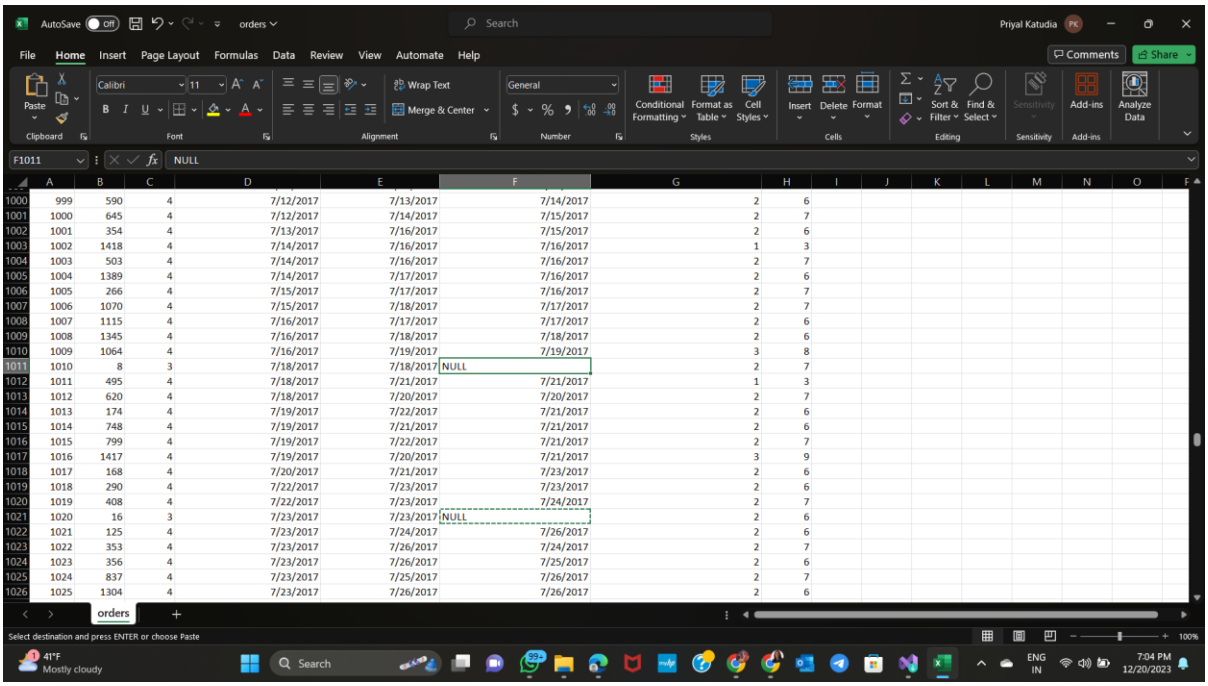
Delete rows with missing values:

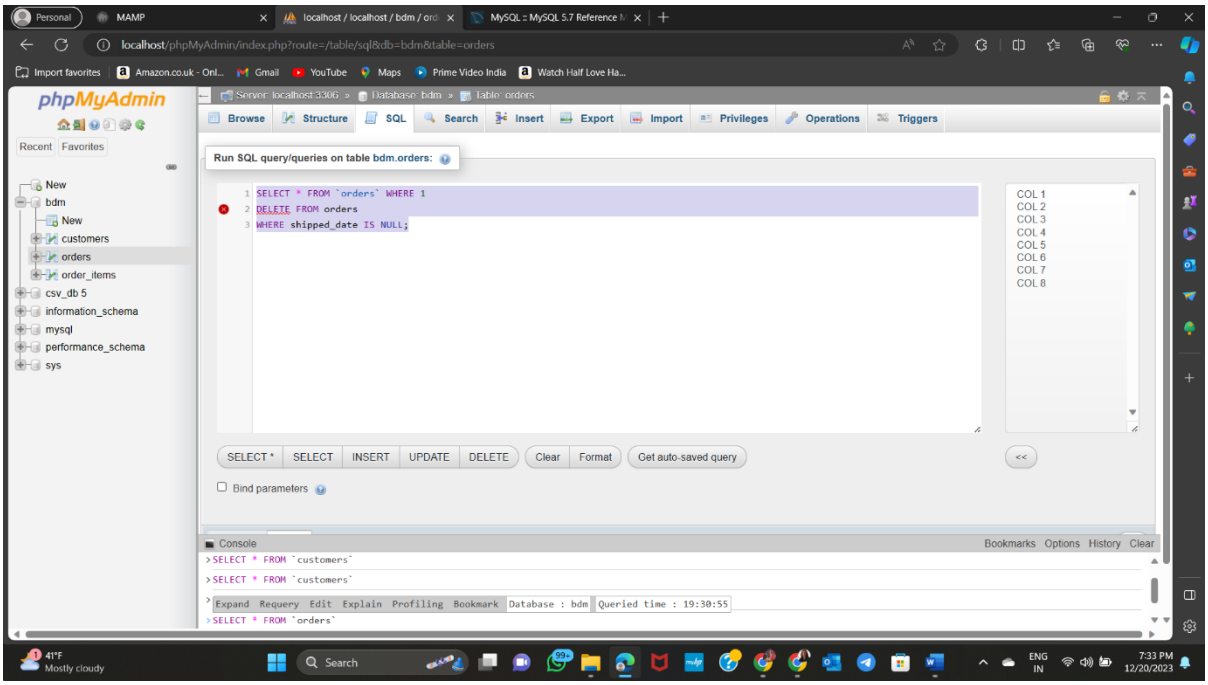
```
DELETE FROM orders
WHERE shipped_date IS NULL;
```

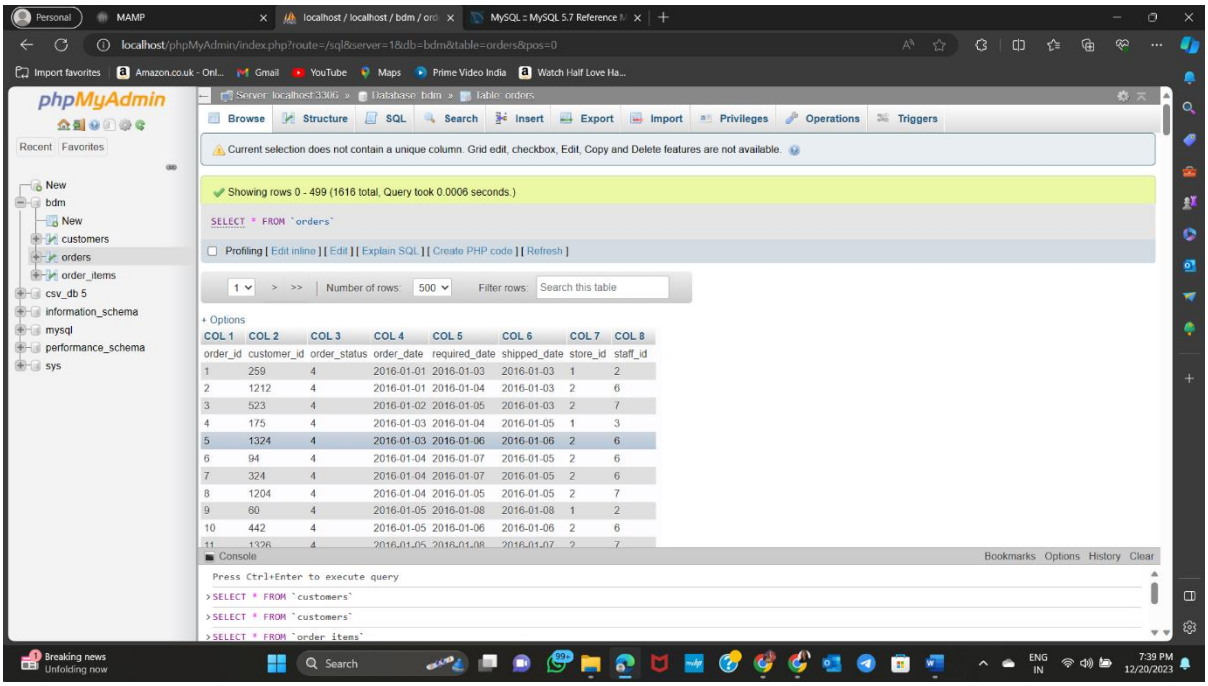


921	628	4	5/31/2017	6/2/2017	6/2/2017	2	6	
922	656	4	5/31/2017	6/1/2017	6/3/2017	2	6	
923	769	4	5/31/2017	6/3/2017	6/2/2017	2	7	
924	68	3	6/2/2017	6/2/2017	6/3/2017	3	8	
925	1316	4	6/2/2017	6/5/2017	6/3/2017	2	7	
926	233	4	6/3/2017	6/5/2017	6/6/2017	1	2	
927	1438	4	6/3/2017	6/5/2017	6/6/2017	1	2	
928	957	4	6/5/2017	6/6/2017	6/6/2017	1	3	
929	472	4	6/5/2017	6/6/2017	6/8/2017	2	6	
930	1224	4	6/5/2017	6/6/2017	6/6/2017	3	9	
931	631	4	6/7/2017	6/8/2017	6/8/2017	2	6	
932	804	4	6/7/2017	6/8/2017	6/9/2017	2	6	
933	909	4	6/7/2017	6/10/2017	6/8/2017	2	6	
934	93	4	6/9/2017	6/10/2017	6/12/2017	2	7	
935	43	3	6/10/2017	6/10/2017	NULL	3	8	
936	265	4	6/10/2017	6/13/2017	6/12/2017	2	6	
937	73	4	6/11/2017	6/14/2017	6/13/2017	2	7	
938	129	4	6/11/2017	6/12/2017	6/12/2017	2	6	
939	1407	4	6/11/2017	6/12/2017	6/14/2017	2	6	
940	585	4	6/12/2017	6/14/2017	6/13/2017	1	2	
941	736	4	6/12/2017	6/14/2017	6/14/2017	1	2	
942	537	4	6/13/2017	6/16/2017	6/15/2017	2	6	
943	1025	4	6/13/2017	6/15/2017	6/15/2017	2	6	
944	138	4	6/14/2017	6/15/2017	6/17/2017	1	3	
945	167	4	6/14/2017	6/15/2017	6/16/2017	2	7	
946	525	4	6/14/2017	6/16/2017	6/15/2017	2	6	
947	278	4	6/15/2017	6/18/2017	6/17/2017	1	3	


phpMyAdmin
Server: localhost:3306 » Database: bdm » Table: orders
Browse Structure SQL Search Insert Export Import Privileges Operations Triggers
Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available.
Showing rows 0 - 499 (1616 total, Query took 0.0012 seconds.)
SELECT * FROM `orders`
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Number of rows: 500 Filter rows: Search this table
+ Options
COL 1 COL 2 COL 3 COL 4 COL 5 COL 6 COL 7 COL 8
order_id customer_id order_status order_date required_date shipped_date store_i staff_id
1 259 4 2016-01-01 2016-01-03 2016-01-03 1 2
2 1212 4 2016-01-01 2016-01-04 2016-01-03 2 6
3 523 4 2016-01-02 2016-01-05 2016-01-03 2 7
4 175 4 2016-01-03 2016-01-04 2016-01-05 1 3
5 1324 4 2016-01-03 2016-01-06 2016-01-06 2 6
6 94 4 2016-01-04 2016-01-07 2016-01-05 2 6
7 324 4 2016-01-04 2016-01-07 2016-01-05 2 6
8 1204 4 2016-01-04 2016-01-05 2016-01-05 2 7
9 60 4 2016-01-05 2016-01-08 2016-01-08 1 2
10 442 4 2016-01-05 2016-01-06 2016-01-06 2 6
Console
Press Ctrl+Enter to execute query
>SELECT * FROM `customers`
>SELECT * FROM `customers`
>SELECT * FROM `order_items`


orders
F1011 NULL
1000 999 590 4 7/12/2017 7/13/2017 7/14/2017 2 6
1001 1000 645 4 7/12/2017 7/14/2017 7/15/2017 2 7
1002 1001 354 4 7/13/2017 7/16/2017 7/15/2017 2 6
1003 1002 1418 4 7/14/2017 7/16/2017 7/16/2017 1 3
1004 1003 503 4 7/14/2017 7/16/2017 7/16/2017 2 7
1005 1004 1389 4 7/14/2017 7/17/2017 7/16/2017 2 6
1006 1005 266 4 7/15/2017 7/17/2017 7/16/2017 2 7
1007 1006 1070 4 7/15/2017 7/18/2017 7/17/2017 2 7
1008 1007 1115 4 7/16/2017 7/17/2017 7/17/2017 2 6
1009 1008 1345 4 7/16/2017 7/18/2017 7/18/2017 2 6
1010 1009 1064 4 7/16/2017 7/19/2017 7/19/2017 3 8
1011 1010 8 3 7/18/2017 7/18/2017 NULL 2 7
1012 1011 495 4 7/18/2017 7/21/2017 7/21/2017 1 3
1013 1012 620 4 7/18/2017 7/20/2017 7/20/2017 2 7
1014 1013 174 4 7/19/2017 7/22/2017 7/21/2017 2 6
1015 1014 748 4 7/19/2017 7/21/2017 7/21/2017 2 6
1016 1015 799 4 7/19/2017 7/22/2017 7/21/2017 2 7
1017 1016 1417 4 7/19/2017 7/20/2017 7/21/2017 3 9
1018 1017 168 4 7/20/2017 7/21/2017 7/23/2017 2 6
1019 1018 290 4 7/22/2017 7/23/2017 7/23/2017 2 6
1020 1019 408 4 7/22/2017 7/23/2017 7/24/2017 2 7
1021 1020 16 3 7/23/2017 7/23/2017 NULL 2 6
1022 1021 125 4 7/23/2017 7/24/2017 7/26/2017 2 6
1023 1022 353 4 7/23/2017 7/26/2017 7/24/2017 2 7
1024 1023 356 4 7/23/2017 7/26/2017 7/25/2017 2 6
1025 1024 837 4 7/23/2017 7/25/2017 7/26/2017 2 7
1026 1025 1304 4 7/23/2017 7/26/2017 7/26/2017 2 6





Deal with outliers:

Identify outliers:

SELECT * FROM your_table
WHERE column_name > upper_bound OR column_name < lower_bound;

CREATE SQL CODE AND IMPORT DATASET:

Use Sql workshop> utilities> data workshop> Load function on apex oracle to upload csv files

Then used insert statements to insert data from uploaded tables into created tables in mySQL

1) BRANDS

```sql
CREATE TABLE brands (
    brand_id INT PRIMARY KEY,
    brand_name VARCHAR(50) NOT NULL
);
```

	A	B	C
1	brand_id	brand_name	
2	1	Electra	
3	2	Haro	
4	3	Heller	
5	4	Pure Cycles	
6	5	Ritchey	
7	6	Strider	
8	7	Sun Bicycles	
9	8	Surly	
10	9	Trek	

Table has brand_id as primary key and columns giving information about different brands.

2) Categories:

```sql
CREATE TABLE categories (
    category_id INT PRIMARY KEY,
    category_name VARCHAR(50) NOT NULL
);
```

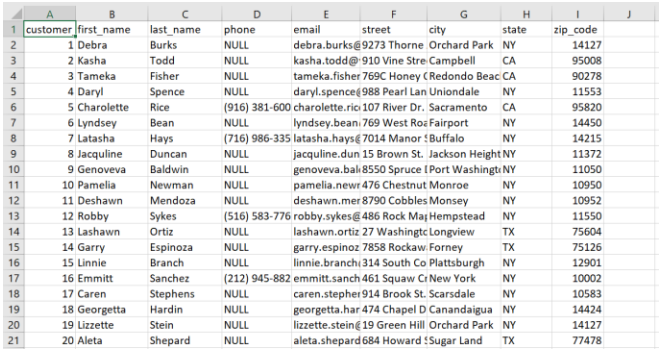
	A	B	C
1	category_id	category_name	
2	1	Children Bicycles	
3	2	Comfort Bicycles	
4	3	Cruisers Bicycles	
5	4	Cyclocross Bicycles	
6	5	Electric Bikes	
7	6	Mountain Bikes	
8	7	Road Bikes	
9			

Table has category_id as primary key and columns giving information about different categories.

3) Customers

```sql
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50),
    phone VARCHAR(20),
    email VARCHAR(100),
    street VARCHAR(100),
    city VARCHAR(50),
    state VARCHAR(2),
    zip_code VARCHAR(10)
);

```



customer	first_name	last_name	phone	email	street	city	state	zip_code
1	Debra	Burks	NULL	debra.burks@	9273 Thorne	Orchard Park	NY	14127
2	Kasha	Todd	NULL	kasha.todd@	910 Vine Stre	Campbell	CA	95008
3	Tameka	Fisher	NULL	tameka.fisher	769C Honey (	Redondo Beac	CA	90278
4	Daryl	Spence	NULL	daryl.spence@	988 Pearl Lan	Uniondale	NY	11553
5	Charolette	Rice	(916) 381-600	charolette.ric	107 River Dr.	Sacramento	CA	95820
6	Lyndsey	Bean	NULL	lyndsey.bean	769 West Roa	Fairport	NY	14450
7	Latasha	Hays	(716) 986-335	latasha.hays@	7014 Manor S	Buffalo	NY	14215
8	Jacquline	Duncan	NULL	jacquline.dun	15 Brown St.	Jackson Height	NY	11372
9	Genoveva	Baldwin	NULL	genoveva.bal	8550 Spruce I	Port Washingt	NY	11050
10	Pamelia	Newman	NULL	pamelia.newr	476 Chestnut	Monroe	NY	10950
11	Deshawn	Mendoza	NULL	deshawn.mer	8790 Cobbles	Monsey	NY	10952
12	Robby	Sykes	(516) 583-776	robby.sykes@	486 Rock Ma	Hempstead	NY	11550
13	Lashawn	Ortiz	NULL	lashawn.ortiz	27 Washingtc	Longview	TX	75604
14	Garry	Espinoza	NULL	garry.espinoz	7858 Rockaw	Forney	TX	75126
15	Linnie	Branch	NULL	linnie.branch	314 South Co	Plattsburgh	NY	12901
16	Emmitt	Sanchez	(212) 945-882	emmitt.sanch	461 Squaw Cr	New York	NY	10002
17	Caren	Stephens	NULL	caren.stepher	914 Brook St.	Scarsdale	NY	10583
18	Georgetta	Hardin	NULL	georgetta.har	474 Chapel D	Canandaigua	NY	14424
19	Lizzette	Stein	NULL	lizzette.stein@	19 Green Hill	Orchard Park	NY	14127
20	Aleta	Shepard	NULL	aleta.shepard	684 Howard S	Sugar Land	TX	77478

Table has customer_id as primary key and columns giving information about contact and address of customers.

4) Order_items

```sql
CREATE TABLE order_items (
    order_id INT,
    item_id INT,
    product_id INT,
    quantity INT,
    list_price DECIMAL(10, 2),
    discount DECIMAL(4, 2),
    PRIMARY KEY (order_id, item_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);

```

order_id	item_id	product_id	quantity	list_price	discount
1	1	20	1	599.99	0.2
1	2	8	2	1799.99	0.07
1	3	10	2	1549	0.05
1	4	16	2	599.99	0.05
1	5	4	1	2899.99	0.2
2	1	20	1	599.99	0.07
2	2	16	2	599.99	0.05
3	1	3	1	999.99	0.05
3	2	20	1	599.99	0.05
4	1	2	2	749.99	0.1
5	1	10	2	1549	0.05
5	2	17	1	429	0.07
5	3	26	1	599.99	0.07
6	1	18	1	449	0.07
6	2	12	2	549.99	0.05
6	3	20	1	599.99	0.1
6	4	3	2	999.99	0.07
6	5	9	2	2999.99	0.07
7	1	15	1	529.99	0.07
7	2	3	1	999.99	0.1

Table has primary key(order_id, item_id) and columns giving information about product and prices.

5) Orders

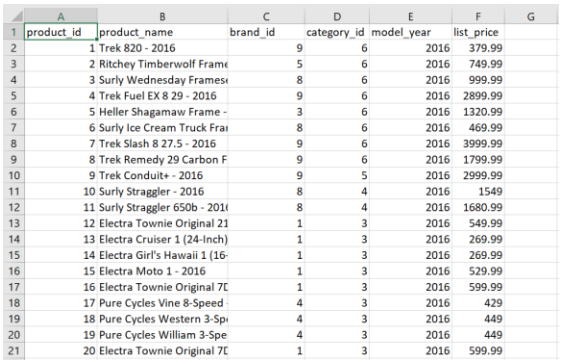
```sql
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_status INT,
    order_date DATE,
    required_date DATE,
    shipped_date DATE,
    store_id INT,
    staff_id INT,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
    FOREIGN KEY (store_id) REFERENCES stores(store_id),
    FOREIGN KEY (staff_id) REFERENCES staff(staff_id)
);
```

order_id	customer_id	order_status	order_date	required_date	shipped_date	store_id	staff_id
1	259	4	1/1/2016	1/3/2016	1/3/2016	1	2
2	1212	4	1/1/2016	1/4/2016	1/3/2016	2	6
3	523	4	1/2/2016	1/5/2016	1/3/2016	2	7
4	175	4	1/3/2016	1/4/2016	1/5/2016	1	3
5	1324	4	1/3/2016	1/6/2016	1/6/2016	2	6
6	94	4	1/4/2016	1/7/2016	1/5/2016	2	6
7	324	4	1/4/2016	1/7/2016	1/5/2016	2	6
8	1204	4	1/4/2016	1/5/2016	1/5/2016	2	7
9	60	4	1/5/2016	1/8/2016	1/8/2016	1	2
10	442	4	1/5/2016	1/6/2016	1/6/2016	2	6
11	1326	4	1/5/2016	1/8/2016	1/7/2016	2	7
12	91	4	1/6/2016	1/8/2016	1/9/2016	1	2
13	873	4	1/8/2016	1/11/2016	1/11/2016	2	6
14	258	4	1/9/2016	1/11/2016	1/12/2016	1	3
15	450	4	1/9/2016	1/10/2016	1/12/2016	2	7
16	552	4	1/12/2016	1/15/2016	1/15/2016	1	3
17	1175	4	1/12/2016	1/14/2016	1/14/2016	1	3
18	541	4	1/14/2016	1/17/2016	1/15/2016	1	3
19	696	4	1/14/2016	1/17/2016	1/16/2016	1	2
20	923	4	1/14/2016	1/16/2016	1/17/2016	1	2

Table has order_id as primary key and columns giving information about Orders.

6) Products

```sql
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(255) NOT NULL,
    brand_id INT,
    category_id INT,
    model_year INT,
    list_price DECIMAL(10, 2),
    FOREIGN KEY (brand_id) REFERENCES brands(brand_id),
    FOREIGN KEY (category_id) REFERENCES categories(category_id)
);
```



product_id	product_name	brand_id	category_id	model_year	list_price
1	Trek 820 - 2016	9	6	2016	379.99
2	Ritchey Timberwolf Frame	5	6	2016	749.99
3	Surly Wednesday Frames	8	6	2016	999.99
4	Trek Fuel EX 8 29 - 2016	9	6	2016	2899.99
5	Heller Shagamaw Frame -	3	6	2016	1320.99
6	Surly Ice Cream Truck Fra	8	6	2016	469.99
7	Trek Slash 8 27.5 - 2016	9	6	2016	3999.99
8	Trek Remedy 29 Carbon F	9	6	2016	1799.99
9	Trek Conduit+ - 2016	9	5	2016	2999.99
10	Surly Straggler - 2016	8	4	2016	1549
11	Surly Straggler 650b - 201	8	4	2016	1680.99
12	Electra Townie Original 21	1	3	2016	549.99
13	Electra Cruiser 1 (24-Inch)	1	3	2016	269.99
14	Electra Girl's Hawaii 1 (16-	1	3	2016	269.99
15	Electra Moto 1 - 2016	1	3	2016	529.99
16	Electra Townie Original 7[	1	3	2016	599.99
17	Pure Cycles Vine 8-Speed	4	3	2016	429
18	Pure Cycles Western 3-Sp	4	3	2016	449
19	Pure Cycles William 3-Spe	4	3	2016	449
20	Electra Townie Original 7[	1	3	2016	599.99

Table has product_id as primary key and columns giving information about products.

7) Staffs

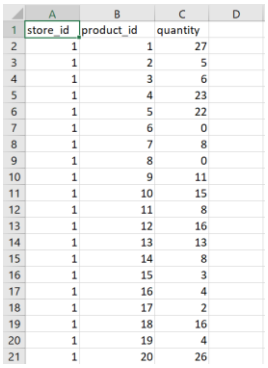
```sql
CREATE TABLE staff (
    staff_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL,
    phone VARCHAR(20) NOT NULL,
    active INT NOT NULL,
    store_id INT,
    manager_id INT,
    FOREIGN KEY (store_id) REFERENCES stores(store_id),
    FOREIGN KEY (manager_id) REFERENCES staff(staff_id)
);
```

staff_id	first_name	last_name	email	phone	active	store_id	manager_id
1	Fabiola	Jackson	fabiola.jackson	(831) 555-5554	1	1	NULL
2	Mireya	Copeland	mireya.copelar	(831) 555-5555	1	1	1
3	Genna	Serrano	genna.serrano	(831) 555-5556	1	1	2
4	Virgie	Wiggins	virgie.wiggins@	(831) 555-5557	1	1	2
5	Jannette	David	jannette.david	(516) 379-4444	1	2	1
6	Marcelene	Boyer	marcelene.boy	(516) 379-4445	1	2	5
7	Venita	Daniel	venita.daniel@	(516) 379-4446	1	2	5
8	Kali	Vargas	kali.vargas@bil	(972) 530-5555	1	3	1
9	Layla	Terrell	layla.terrell@b	(972) 530-5556	1	3	7
10	Bernardine	Houston	bernardine.ho	(972) 530-5557	1	3	7

Table has staff_id as primary key and columns giving information about staff members.

8) Stocks

```sql
CREATE TABLE stocks (
    store_id INT,
    product_id INT,
    quantity INT,
    PRIMARY KEY (store_id, product_id),
    FOREIGN KEY (store_id) REFERENCES stores(store_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```



store_id	product_id	quantity
1	1	27
1	2	5
1	3	6
1	4	23
1	5	22
1	6	0
1	7	8
1	8	0
1	9	11
1	10	15
1	11	8
1	12	16
1	13	13
1	14	8
1	15	3
1	16	4
1	17	2
1	18	16
1	19	4
1	20	26

Table has primary key(store_id, product_id) and columns giving information about stocks of products.

9) Stores

```sql
CREATE TABLE stores (
    store_id INT PRIMARY KEY,
    store_name VARCHAR(100) NOT NULL,
    phone VARCHAR(20) NOT NULL,
    email VARCHAR(100) NOT NULL,
    street VARCHAR(255) NOT NULL,
    city VARCHAR(50) NOT NULL,
    state VARCHAR(2) NOT NULL,
    zip_code VARCHAR(10) NOT NULL
);
```

store_id	store_name	phone	email	street	city	state	zip_code
1	Santa Cruz Bikes	(831) 476-4321	santacruz@bikes.sl	3700 Portola Drive	Santa Cruz	CA	95060
2	Baldwin Bikes	(516) 379-8888	baldwin@bikes.sho	4200 Chestnut Lan	Baldwin	NY	11432
3	Rowlett Bikes	(972) 530-5555	rowlett@bikes.sho	8000 Fairway Aver	Rowlett	TX	75088

Table has store_id as primary key and columns giving information about different stores.

Queries:

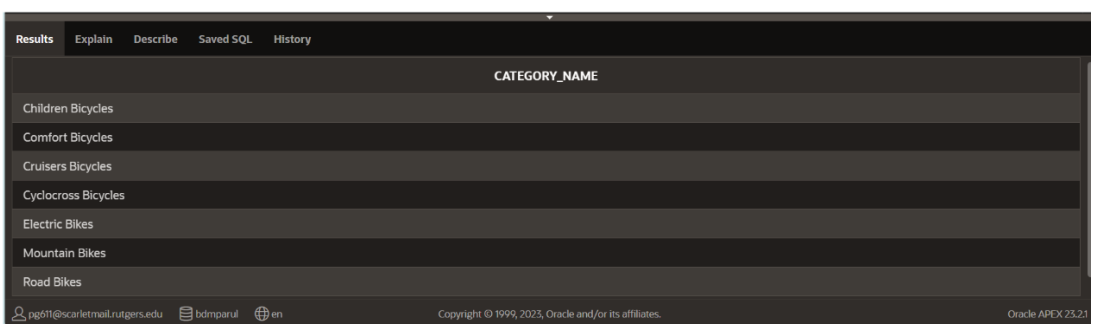
QUERY 1:

select category_name

From CATEGORIES

```
1   select category_name
2   From CATEGORIES
```

Output:



Explanation:

- It provides all the categories of bike from category table.

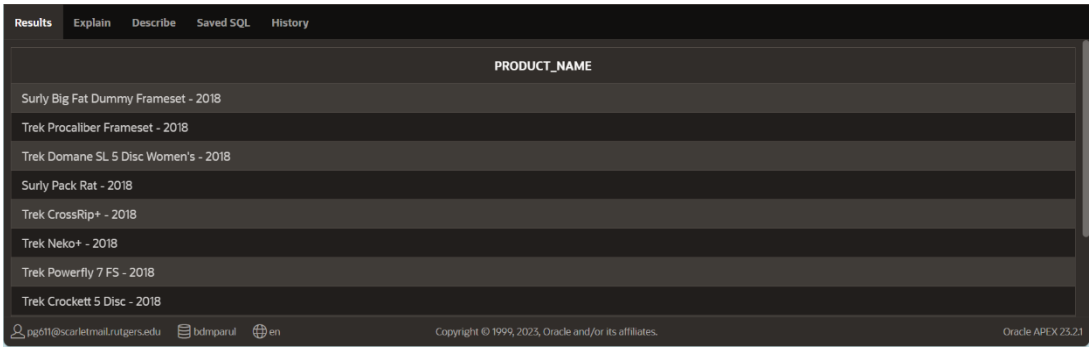
QUERY 2:

SELECT DISTINCT Product_Name

FROM PRODUCTS

```
1   SELECT DISTINCT Product_Name
2   FROM PRODUCTS
```

Output:



Explanation:

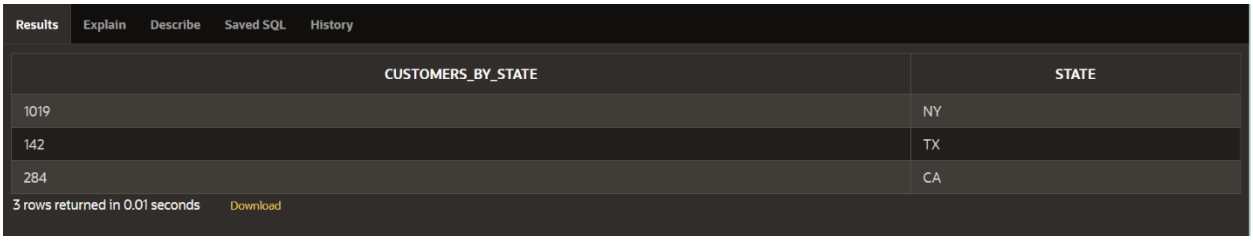
- It provides Unique product names of bike from products table.

QUERY 3:

SELECT count(*) as customers_by_state,state
FROM customers
GROUP BY state;

```
1   SELECT count(*) as customers_by_state,state
2   FROM customers
3   GROUP BY state;
```

Output:



Explanation:

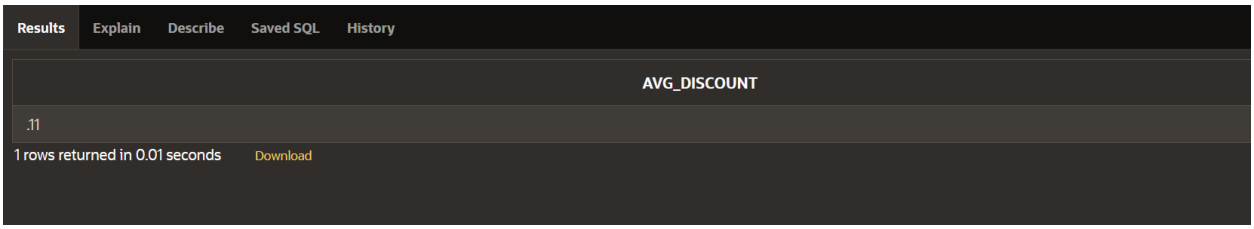
- This counts the total number of customers in each state.

QUERY 4:

SELECT ROUND(AVG(discount),2) as avg_discount
FROM ORDER_ITEMS;

```
1   SELECT ROUND(AVG(discount),2) as avg_discount
2   FROM ORDER_ITEMS;
```

Output:



Explanation:

It is essential to get the information of what average discounts the customers are getting.

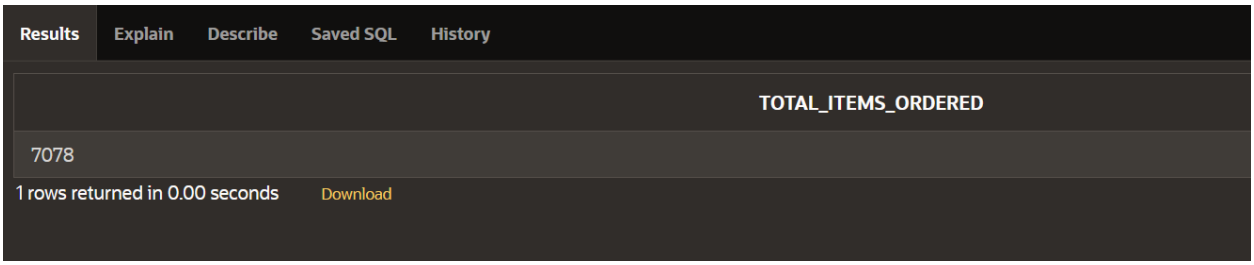
The ROUND function in SQL is used to round a numeric value to a specified number of decimal places.

QUERY 5:

SELECT SUM(quantity) AS total_items_ordered
FROM ORDER_ITEMS;

```
1   SELECT SUM(quantity) AS total_items_ordered
2   FROM ORDER_ITEMS;
```

Output:



Explanation:

Use sum function to get total items quantity that is sold.

QUERY 6:

SELECT product_id,discount
FROM ORDER_ITEMS
WHERE discount <= 0.2
ORDER BY discount DESC;

```
1   SELECT product_id,discount
2   FROM ORDER_ITEMS
3   WHERE discount <= 0.2
4   ORDER BY discount DESC;
```

Output:

Results | Explain | Describe | Saved SQL | History

PRODUCT_ID	DISCOUNT
2	.2
4	.2
11	.2
10	.2
8	.2
20	.2
6	.2
2	.2

Explanation:

- This is important because it will indicate what all products have the low discount to buy.

QUERY 7:

SELECT ROUND(AVG(list_price * (1 - discount)),2) AS avg_SALEPRICE_usd
FROM ORDER_ITEMS
WHERE list_price >=1000;

```
1   SELECT ROUND(AVG(list_price * ( 1 - discount)),2) AS avg_SALEPRICE_usd
2   FROM ORDER_ITEMS
3   WHERE list_price >=1000;
```

Output:

Results | Explain | Describe | Saved SQL | History

AVG_SALEPRICE_USD
2542.02

1 rows returned in 0.01 seconds Download

Explanation:

- This information is important because it will indicate the value of sales more than 1000.

QUERY 8-

SELECT *
FROM CUSTOMERS
WHERE state IN ('CA', 'NY')

```
1   SELECT *
2   FROM CUSTOMERS
3   WHERE state IN ('CA', 'NY')
```

Output:

Results | Explain | Describe | Saved SQL | History

CUSTOMER_ID	FIRST_NAME	LAST_NAME	PHONE	EMAIL	STREET	CITY	STATE	ZIP_CODE
79	Ashanti	Parks	NULL	ashanti.parks@hotmail.com	846 N. Helen St.	Baldwin	NY	11510
80	Sarai	Mckee	(716) 912-8110	sarai.mckee@msn.com	641 Glenwood Avenue	Buffalo	NY	14215
81	Zina	Bonner	NULL	zina.bonner@hotmail.com	9312 8th Street	San Lorenzo	CA	94580
82	Lizzie	Joyner	NULL	lizzie.joyner@msn.com	8541 Roberts St.	Coachella	CA	92236
83	Tiesha	Daniel	NULL	tiesha.daniel@yahoo.com	6 West Bohemia Lane	Scarsdale	NY	10583
84	Armand	Whitehead	NULL	armand.whitehead@hotmail.com	639 Harvey St.	Lindenhurst	NY	11757
85	Teofila	Fischer	NULL	teofila.fischer@aol.com	1 West Brickyard St.	Huntington Station	NY	11746
86	Lissa	Vargas	NULL	lissa.vargas@yahoo.com	517 Victoria Ave.	Oswego	NY	13126

pg611@scarletmail.rutgers.edu bdmparul en Oracle APEX 23.2.1

Explanation: This SQL query selects all columns from the "CUSTOMERS" table where the state is either California or New York, filtering the results based on the specified conditions.

QUERY 9–

SELECT * FROM PRODUCTS

WHERE product_name LIKE 'T%'

```
1  SELECT *
2  FROM PRODUCTS
3  WHERE product_name LIKE 'T%'
```

Results | Explain | Describe | Saved SQL | History

ID	PRODUCT_ID	PRODUCT_NAME	BRAND_ID	CATEGORY_ID	MODEL_YEAR	LIST_PRICE
125	125	Trek Kids' Dual Sport - 2018	9	6	2018	469.99
129	129	Trek X-Caliber 7 - 2018	9	6	2018	919.99
130	130	Trek Stache Carbon Frameset - 2018	9	6	2018	919.99
132	132	Trek Procal AL Frameset - 2018	9	6	2018	1499.99
133	133	Trek Procaliber Frameset - 2018	9	6	2018	1499.99
134	134	Trek Remedy 27.5 C Frameset - 2018	9	6	2018	1499.99
135	135	Trek X-Caliber Frameset - 2018	9	6	2018	1499.99
136	136	Trek Procaliber 6 - 2018	9	6	2018	1799.99

pg611@scarletmail.rutgers.edu bdmparul en Oracle APEX 23.2.1

Explanation: This SQL query selects all columns from the "PRODUCTS" table where the product name begins with the letter 'T', using the LIKE operator with a pattern match.

QUERY 10 :

SELECT COUNT(*) as sold_amount, staff_id

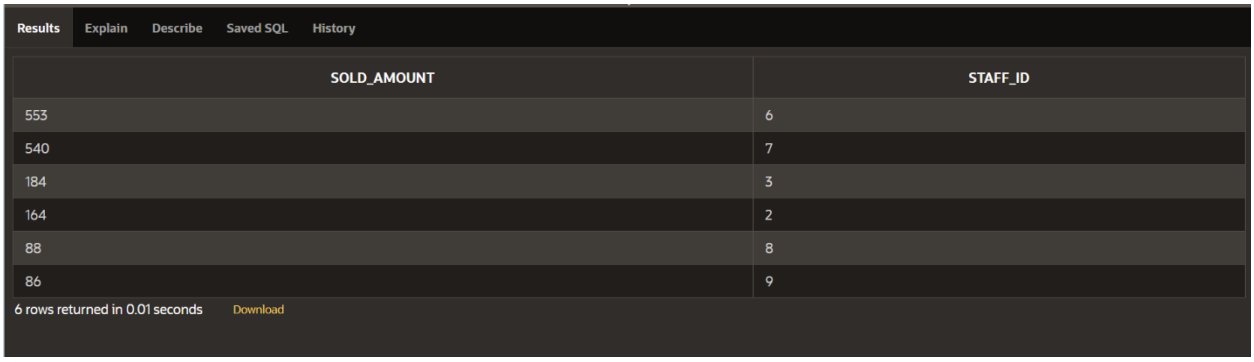
FROM orders

GROUP BY staff_id

ORDER BY sold_amount DESC

```sql
SELECT COUNT(*) as sold_amount, staff_id
FROM orders
GROUP BY staff_id
ORDER BY sold_amount DESC
```

Output:



Results | Explain | Describe | Saved SQL | History

SOLD_AMOUNT	STAFF_ID
553	6
540	7
184	3
164	2
88	8
86	9

6 rows returned in 0.01 seconds Download

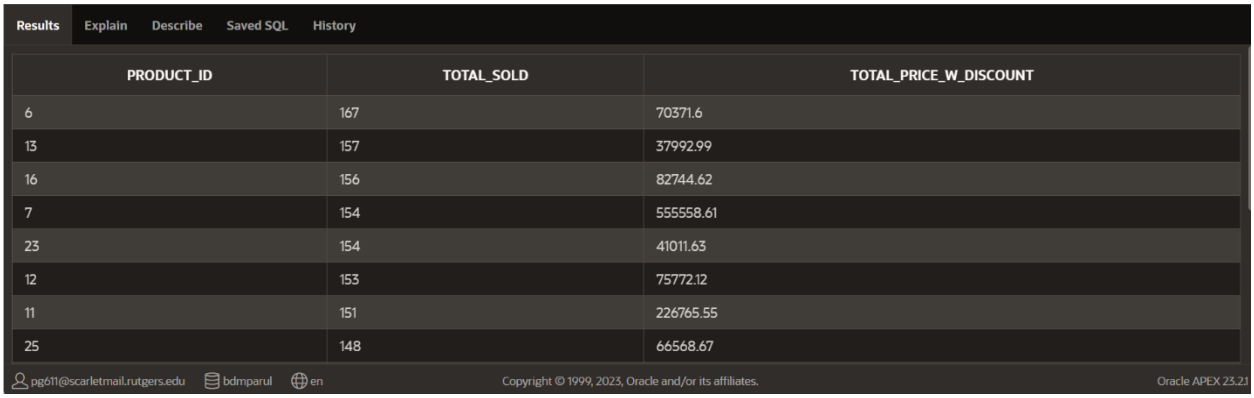
Explanation: This SQL query counts the number of orders handled by each staff member, groups the results by staff ID, and orders the output to display the top staff members based on the count of orders in descending order

QUERY 11 :

SELECT product_id,
SUM(quantity) AS total_sold,
ROUND(SUM(list_price * (1 - discount) * quantity),2) AS total_price_w_discount
FROM ORDER_ITEMS
GROUP BY product_id
ORDER BY total_sold DESC;

```sql
SELECT product_id,
SUM(quantity) AS total_sold,
ROUND(SUM(list_price * (1 - discount) * quantity),2) AS total_price_w_discount
FROM ORDER_ITEMS
GROUP BY product_id
ORDER BY total_sold DESC;
```

Output:



Results | Explain | Describe | Saved SQL | History

PRODUCT_ID	TOTAL_SOLD	TOTAL_PRICE_W_DISCOUNT
6	167	70371.6
13	157	37992.99
16	156	82744.62
7	154	555558.61
23	154	41011.63
12	153	75772.12
11	151	226765.55
25	148	66568.67

pg611@scarletmail.rutgers.edu | bdmparul | en | | Oracle APEX 23.2.1

Explanation: It Provides the hot product id in the market and its total value for sale.

QUERY 12:

```
SELECT
   ord.product_id,
   SUM(ord.quantity) AS quant,
   ord.list_price,
   prod.product_name
FROM
   order_items ord
INNER JOIN
   products prod ON ord.product_id = prod.product_id
GROUP BY
   ord.product_id, ord.list_price, prod.product_name
ORDER BY
   quant DESC;
```

```sql
SELECT
    ord.product_id,
    SUM(ord.quantity) AS quant,
    ord.list_price,
    prod.product_name
FROM
    order_items ord
INNER JOIN
    products prod ON ord.product_id = prod.product_id
GROUP BY
    ord.product_id, ord.list_price, prod.product_name
ORDER BY
    quant DESC;
```

Output:

Results | Explain | Describe | Saved SQL | History

PRODUCT_ID	QUANT	LIST_PRICE	PRODUCT_NAME
6	167	469.99	Surly Ice Cream Truck Frameset - 2016
13	157	269.99	Electra Cruiser 1 (24-Inch) - 2016
16	156	599.99	Electra Townie Original 7D EQ - 2016
23	154	299.99	Electra Girl's Hawaii 1 (20-inch) - 2015/2016
7	154	3999.99	Trek Slash 8 27.5 - 2016
12	153	549.99	Electra Townie Original 21D - 2016
11	151	1680.99	Surly Straggler 650b - 2016
25	148	499.99	Electra Townie Original 7D - 2015/2016

Explanation:

- The query retrieves product details, including the total quantity sold (quant), list price, and product name, by joining the order_items and products tables, grouping by product ID, list price, and product name, and ordering the results by total quantity in descending order.

QUERY 13:

Select
Count(o.order_id) as sold_amount,
o.staff_id, o.store_id, s.first_name, s.last_name
from Orders o
Join staffs s ON o.staff_id = s.staff_id
GROUP BY o.staff_id, o.store_id, s.first_name, s.last_name

```
1   SELECT
2       COUNT(o.order_id) as sold_amount,
3       o.staff_id,
4       o.store_id,
5       s.first_name,
6       s.last_name
7   FROM ORDERS o
8   JOIN STAFFS s ON o.staff_id = s.staff_id
9   GROUP BY o.staff_id, o.store_id, s.first_name, s.last_name
10  ORDER BY sold_amount DESC
11  FETCH FIRST 3 ROWS ONLY;
12
```

Output:

Results | Explain | Describe | Saved SQL | History

SOLD_AMOUNT	STAFF_ID	STORE_ID	FIRST_NAME	LAST_NAME
553	6	2	Marcelene	Boyer
540	7	2	Venita	Daniel
184	3	1	Genna	Serrano

3 rows returned in 0.02 seconds Download

Explanation: It uses the COUNT function to calculate the number of orders handled by each staff member, joins the orders and staffs tables based on the staff ID, groups the results by staff ID, store ID, first name, and last name, and then orders the results in descending order of the order count. The FETCH FIRST 3 ROWS ONLY clause limits the output to the top
3 staff members.

QUERY 14:

```
SELECT
   COUNT(quantity) as quantity_count,
   stores.store_name,
   stores.state
FROM STOCKS
INNER JOIN STORES
ON stocks.store_id = stores.store_id
GROUP BY stores.store_id, stores.store_name, stores.state
```

Output:

Results | Explain | Describe | Saved SQL | History

QUANTITY_COUNT	STORE_NAME	STATE
313	Santa Cruz Bikes	CA
313	Baldwin Bikes	NY
313	Rowlett Bikes	TX

3 rows returned in 0.00 seconds Download

Explanation: This SQL query counts the quantities in the stocks table for each store, joins with the stores table based on the store ID, groups the results by store ID, store name, and state.

QUERY 15:

```
SELECT DISTINCT order_id,customer_id
FROM ORDERS
WHERE order_id IN (
```

```
SELECT DISTINCT order_id
FROM ORDER_ITEMS
INNER JOIN
   products
ON
   order_items.product_id = products.product_id
AND
   brand_id = 9
AND
   discount >= .20
);
```

```
SELECT DISTINCT order_id,customer_id
FROM ORDERS
WHERE order_id IN (
                   SELECT DISTINCT order_id
                   FROM ORDER_ITEMS
                   INNER JOIN
                       products
                   ON
                       order_items.product_id = products.product_id
                   AND
                       brand_id = 9
                   AND
                       discount >= .20
                   );
```

Output:

Results | Explain | Describe | Saved SQL | History

ORDER_ID	CUSTOMER_ID
212	1373
288	780
297	1425
31	1238
951	339
955	880
961	82
641	673

pg611@scarletmail.rutgers.edu | bdmparul | en | | Oracle APEX 23.2.1

Explanation: This query filters records from an unspecified table where the brand ID is 9 and the discount is 20% or higher.

QUERY 16–

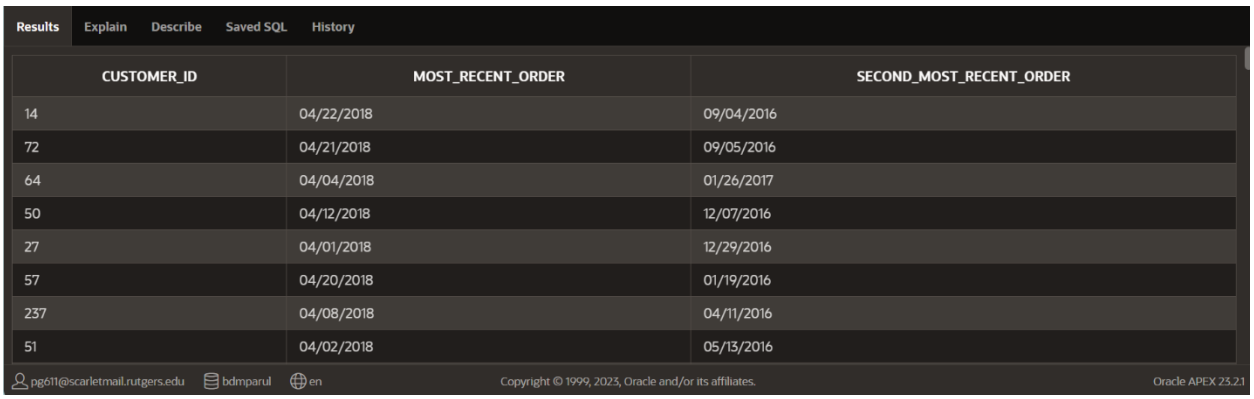
```
SELECT
   t1.customer_id,
   MAX(t1.order_date) AS most_recent_order,
   MAX(t2.order_date) AS second_most_recent_order
```

```
FROM ORDERS t1
INNER JOIN ORDERS t2
ON
   t1.customer_id = t2.customer_id
AND
   t1.order_date > t2.order_date
GROUP BY
   t1.customer_id;
```

```sql
SELECT
    t1.customer_id,
    MAX(t1.order_date) AS most_recent_order,
    MAX(t2.order_date) AS second_most_recent_order
FROM ORDERS t1
INNER JOIN ORDERS t2
ON
    t1.customer_id = t2.customer_id
AND
    t1.order_date > t2.order_date
GROUP BY
    t1.customer_id;

```

Output:



Results | Explain | Describe | Saved SQL | History

CUSTOMER_ID	MOST_RECENT_ORDER	SECOND_MOST_RECENT_ORDER
14	04/22/2018	09/04/2016
72	04/21/2018	09/05/2016
64	04/04/2018	01/26/2017
50	04/12/2018	12/07/2016
27	04/01/2018	12/29/2016
57	04/20/2018	01/19/2016
237	04/08/2018	04/11/2016
51	04/02/2018	05/13/2016

pg611@scarletmail.rutgers.edu | bdmparul | en | Copyright © 1999, 2023, Oracle and/or its affiliates. | Oracle APEX 23.2.1

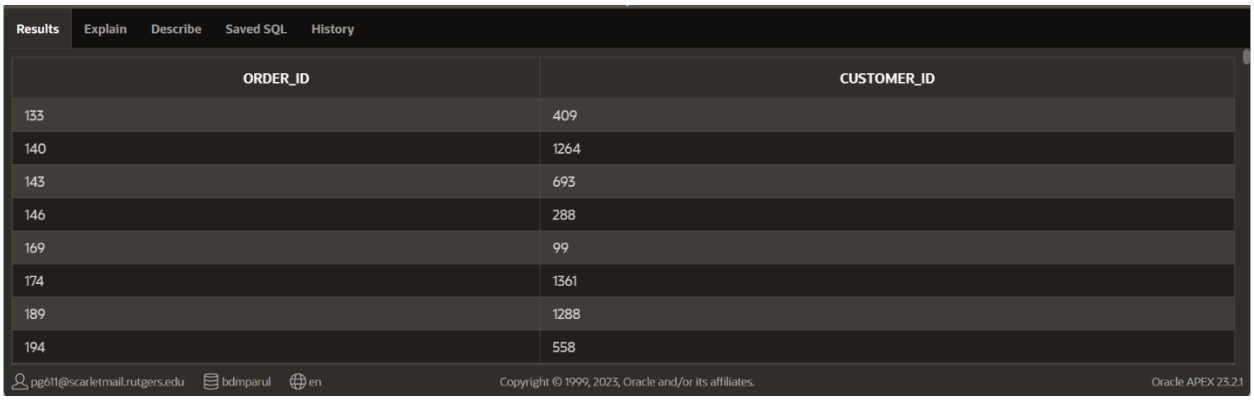
Explanation: This SQL query uses self-joins on the "orders" table to find the most recent and second most recent order dates for each customer, grouping the results by customer ID.

QUERY 17–

```
SELECT DISTINCT order_id, Customer_id
FROM ORDERS
WHERE EXISTS ( SELECT 1 FROM order_items WHERE discount>= 0.20 AND
order_items.order_id = orders.order_id);
```

```sql
SELECT DISTINCT order_id, customer_id
FROM ORDERS
WHERE
    EXISTS (
            SELECT
                1
            FROM
                order_items
            WHERE
                discount >= .20
            AND
                order_items.order_id = orders.order_id
            );
```

Output:



Results | Explain | Describe | Saved SQL | History

ORDER_ID	CUSTOMER_ID
133	409
140	1264
143	693
146	288
169	99
174	1361
189	1288
194	558

pg611@scarletmail.rutgers.edu | bdmparul | en | Copyright © 1999, 2023, Oracle and/or its affiliates. | Oracle APEX 23.2.1

Explanation: This SQL query selects unique order IDs and customer IDs from the "orders" table for orders where at least one item in the corresponding "order_items" table has a discount of 20% or more, utilizing the EXISTS clause for the subquery condition.

QUERY 18–

```sql
SELECT c.category_id, c.category_name, COUNT(p.product_id) AS total_products
FROM categories c
LEFT JOIN products p ON c.category_id = p.category_id
GROUP BY c.category_id, c.category_name;
```

Results | Explain | Describe | Saved SQL | History

CATEGORY_ID	CATEGORY_NAME	TOTAL_PRODUCTS
2	Comfort Bicycles	30
7	Road Bikes	60
1	Children Bicycles	59
5	Electric Bikes	24
3	Cruisers Bicycles	78
6	Mountain Bikes	60
4	Cyclocross Bicycles	10

7 rows returned in 0.04 seconds Download

sb2509@scarletmail.rutgers.edu | bdm.shubhamm | en | Copyright © 1999, 2023, Oracle and/or its affiliates. | Oracle APEX 23.2.1

Explanation: This SQL query retrieves the category ID, category name, and the count of products for each category, including categories with no products. It uses a LEFT JOIN and GROUP BY clause.

Query 19:

```
SELECT SUM(p.list_price * s.quantity) AS total_revenue
FROM stocks s
JOIN products p ON s.product_id = p.product_id
WHERE s.store_id = 1;
```

Results | Explain | Describe | Saved SQL | History

TOTAL_REVENUE
6487242.16

1 rows returned in 0.01 seconds Download

Explanation: This SQL query calculates the total revenue by multiplying the list price of products with their corresponding quantities in stock at store_id 1. It involves a JOIN between "stocks" and "products".

Query 20 :

```
SELECT product_id, quantity
FROM stocks
WHERE store_id = 1 AND quantity > 20;
```

Results | Explain | Describe | Saved SQL | History

PRODUCT_ID	QUANTITY
1	27
4	23
5	22
20	26
21	24
22	29
27	21
30	30

Explanation: This SQL query retrieves the product_id and quantity from the 'sales' table where the store_id is 1 and the quantity is greater than 20. It filters sales data based on specified conditions.

Query21:

```
SELECT state, COUNT(customer_id) AS total_customers
FROM customers
GROUP BY state;
```

Results | Explain | Describe | Saved SQL | History

STATE	TOTAL_CUSTOMERS
NY	1019
TX	142
CA	284

3 rows returned in 0.03 seconds Download

Explanation: This SQL query retrieves the count of customers for each unique state from the "customers" table, grouping the results by state. It provides a summary of the total customers in each state.

Query 22:

```
SELECT b.brand_id, b.brand_name, COUNT(p.product_id) AS product_count
FROM brands b
LEFT JOIN products p ON b.brand_id = p.brand_id
GROUP BY b.brand_id, b.brand_name
ORDER BY product_count DESC
```

Results | Explain | Describe | Saved SQL | History

BRAND_ID	BRAND_NAME	PRODUCT_COUNT
9	Trek	135
1	Electra	118
8	Surly	25
7	Sun Bicycles	23
2	Haro	10
4	Pure Cycles	3
3	Heller	3
6	Strider	3

sb2509@scarletmail.rutgers.edu bdm.shubhamm en Oracle APEX 23.2.1

Explanation: This SQL query retrieves the brand ID, brand name, and the count of products for each brand, including those with no products. It uses a LEFT JOIN, GROUP BY, and orders results by product count in descending order.

Query 23:

```
SELECT
    customer_id,
    COUNT(order_id) AS total_orders
FROM orders
GROUP BY customer_id;
```

Results | Explain | Describe | Saved SQL | History

CUSTOMER_ID	TOTAL_ORDERS
763	1
371	1
237	2
459	1
1127	1
1288	1
1102	1
591	1

Explanation: This SQL query selects the customer ID and counts the total orders placed by each customer from the "orders" table. The results are grouped by customer ID, providing an order count per customer.

Query 24:

```
SELECT *
FROM order_items
WHERE discount > 0.15;
```

Results | Explain | Describe | Saved SQL | History

ID	ORDER_ID	ITEM_ID	PRODUCT_ID	QUANTITY	LIST_PRICE	DISCOUNT
503	179	3	18	1	449	.2
504	179	4	12	2	549.99	.2
509	181	3	15	2	529.99	.2
513	183	1	11	2	1680.99	.2
517	184	1	4	1	2899.99	.2
526	189	1	16	2	599.99	.2
528	190	2	15	1	529.99	.2
529	190	3	3	2	999.99	.2

Explanation: This SQL query selects all columns from the "order_items" table where the

discount is greater than 0.15, filtering and retrieving rows with discounts exceeding the specified threshold.

Query 25:

```
SELECT *
FROM stocks
WHERE quantity = 0;

```

Results | Explain | Describe | Saved SQL | History

ID	STORE_ID	PRODUCT_ID	QUANTITY
6	1	6	0
8	1	8	0
32	1	32	0
42	1	42	0
92	1	92	0
160	1	160	0
163	1	163	0
168	1	168	0

sb2509@scarletmail.rutgers.edu | bdm.shubhamm | en | | Oracle APEX 23.2.1

Explanation: This SQL query selects all columns from the "stocks" table where the quantity is equal to zero, retrieving rows that represent items with no available stock.

Query 26:

```
WITH ranked_products AS (
  SELECT store_id,
         product_id,
         quantity,
         RANK() OVER (PARTITION BY store_id ORDER BY quantity DESC) AS ranking
  FROM stocks
)
SELECT *
FROM ranked_products
WHERE ranking = 1;

```

STORE_ID	PRODUCT_ID	QUANTITY	RANKING
1	30	30	1
1	292	30	1
1	219	30	1
1	193	30	1
1	188	30	1
1	109	30	1
1	106	30	1
1	68	30	1

sb2509@scarletmail.rutgers.edu | bdm.shubhamm | en | Copyright © 1999, 2023, Oracle and/or its affiliates. | Oracle APEX 23.2.1

Explanation: This SQL query uses a common table expression (CTE) to rank products by quantity within each store and selects the products with the highest quantity ranking (1) from the CTE.

Query 27:

```
SELECT product_id,
       MAX(quantity) AS max_quantity,
       MIN(quantity) AS min_quantity,
       AVG(quantity) AS avg_quantity
FROM stocks
GROUP BY product_id;
```

Results | Explain | Describe | Saved SQL | History

PRODUCT_ID	MAX_QUANTITY	MIN_QUANTITY	AVG_QUANTITY
6	27	0	12.666666666666666666666666666666666666667
14	18	4	10
23	12	8	9.666666666666666666666666666666666666667
27	21	10	17.333333333333333333333333333333333333
50	29	7	20.333333333333333333333333333333333333
51	8	2	4
52	23	8	16.333333333333333333333333333333333333
57	27	1	12.333333333333333333333333333333333333

sb2509@scarletmail.rutgers.edu | bdm.shubhamm | en | Copyright © 1999, 2023, Oracle and/or its affiliates. | Oracle APEX 23.2.1

Explanation: This SQL query retrieves the maximum, minimum, and average quantity values for each unique product_id from the "stores" table, grouping the results by product_id.

Query 28:

```
SELECT active, COUNT(*) AS staff_count
FROM staffs
GROUP BY active;
```

Results | Explain | Describe | Saved SQL | History

ACTIVE	STAFF_COUNT
1	10

1 rows returned in 0.01 seconds | Download

Explanation: This SQL query counts the number of staff members for each "active" status, grouping results by the "active" column. It provides a count of active and inactive staff members.

Query 29:

```
SELECT o.order_id, SUM(oi.quantity * oi.list_price * (1 - oi.discount)) AS total_price
FROM order_items o
JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY o.order_id;
```

Results | Explain | Describe | Saved SQL | History

ORDER_ID	TOTAL_PRICE
210	1366.1494
226	712.4905
232	31869.6576
237	9861.8288
277	6593.85
281	24531.1623
290	12285.0414
291	2790.3248

Explanation: This SQL query calculates the total price for each order by multiplying the quantity, list price, and discount-adjusted price of items, grouping results by order ID. It involves joining two tables.

Query 30:

```
SELECT product_id, SUM(quantity) AS total_quantity_sold
FROM stocks
WHERE store_id = 1
GROUP BY product_id
ORDER BY total_quantity_sold DESC
LIMIT 10;
```

Results | Explain | Describe | Saved SQL | History

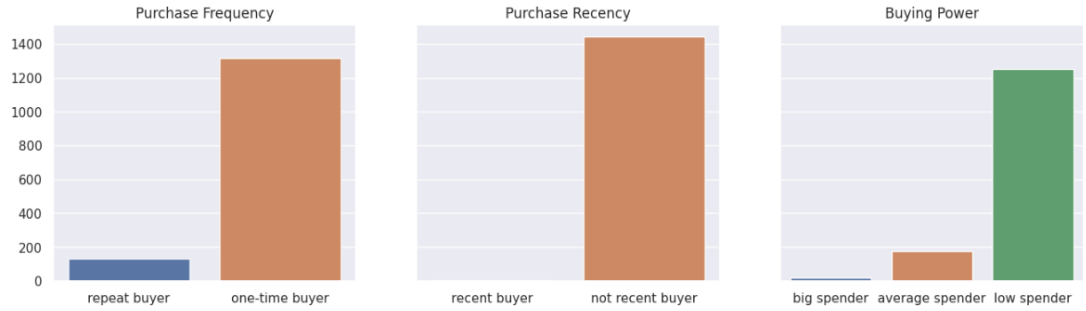
PRODUCT_ID	TOTAL_QUANTITY_SOLD
64	30
68	30
109	30
30	30
106	30
219	30
292	30
193	30

Explanation: This SQL query retrieves the total quantity sold for each product in Store 1, groups results by product, orders them by descending total quantity sold, and limits the output to 10 rows.

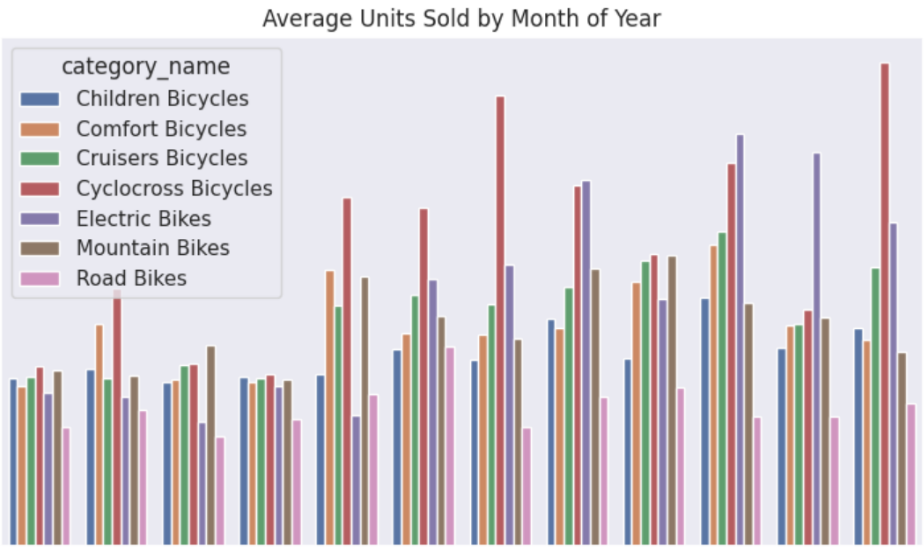
Visualization:

Create a line plot showing the 30-day moving average of orders over time, with each store's trend distinguished by color, while removing axis labels and ticks for a cleaner visualization.

Visualize customer characteristics using subplots with count plots for purchase frequency, purchase recency, and buying power, while sharing the y-axis for easier comparison. Remove axis labels and titles for a clean presentation.



Generate a bar plot illustrating the average units sold per month, differentiated by product category, with axis labels removed for clarity, and displaying the error bars.


Average Units Sold by Month of Year
category_name
Children Bicycles
Comfort Bicycles
Cruisers Bicycles
Cyclocross Bicycles
Electric Bikes
Mountain Bikes
Road Bikes